Natural Language Processing

Fall 2024

Project 1: Conventional Text Categorization Project

For this project, you will implement a text categorization system, using one of the three conventional machine learning methods for text categorization that we covered in class. You must implement the crux of the algorithm yourself. You are allowed to use available NLP resources that are not specifically related to text categorization, machine learning, or word statistics. For example, you may use an existing tokenizer, stemmer, or lemmatizer, if you wish. Assuming you implement the project in Python (which I recommend), you may use NLTK. However, you may not use any pre-existing routine (from NLTK or any other library) that calculates word statistics or that applies text categorization or a machine learning approach.

Your program must allow the user to specify the names of two input files. The first will contain a list of labeled training documents. Each row of the file will list the relative path and filename of one training document followed by a single space and then the category of the document. The system should use these training documents to train itself appropriately so that it can predict the labels of future documents according to the learned categories. The second file will contain a list of unlabeled test documents. Each row will consist of a single string representing the relative path and filename of one test document. The program should loop through the indicated test documents and categorize each one based on its training. After all the predictions have been made, the program should allow the user to specify the name of an output file. This file will list the test documents along with their predicted labels; the file format should be same as the format of the training file. *Do not assume any specific file names, folder names, or relative paths.* It's OK if you assume that the input files (but not the documents) will exist in the current directory.

If you wish, you may separate training and testing into two separate programs. A disadvantage of this approach would be that your training program will need to save all its learned statistics in a file, and the test program would have to reload this information. (If you choose to do this, you should have both programs prompt the user for the name of this file.) An advantage of this approach is that this would allow you to train your system once for a set of categories, and then using the saved representation of your trained system, you could evaluate the trained system on multiple test sets without having to retrain it. Since training is usually more computationally intensive than testing, this can be very beneficial in practice; but for this project, it is optional.

To test your programs, I am providing you with one entire corpus, including a training set and a test set. This corpus involves the categorization of news documents into the categories: `Str` (*Struggle*), `Pol` (*Politics*), `Dis` (*Disaster*), `Cri` (*Crime*), or `Oth` (*Other*). It is guaranteed that each document belongs to exactly one of these categories (i.e., the categories are mutually exclusive and exhaustive). I am also providing you with just the training sets (but not the test sets) for two other corpora. The second corpus involves the categorization of images based on the first sentences of their captions into the mutually exclusive and exhaustive categories: `O`

(*Outdoor*) or `I` (*Indoor*). The third corpus involves the categorization of news documents into the mutually exclusive and exhaustive categories: `Wor` (*World News*), `USN` (*U.S. News*), `Sci` (*Science and Technology*), `Fin` (*Finance*), `Spo` (*Sports*), or `Ent` (*Entertainment*).

Note that *I am only providing the training sets for two of the corpora*. (I have separate test sets for these corpora that I will use for my evaluation.) Since I am not providing these test sets, you should use one of the methods mentioned in class to evaluate your system for these corpora; e.g., you can either split the training set into a smaller training set and a tuning set, or you can apply k-fold cross-validation. You should never include any documents in both the training set and the tuning set for a single experiment; that would likely lead to an unreasonably high estimate of future accuracy. Generally, using either a tuning set or cross-validation should give you a reasonable idea as to what sort of accuracy to expect when I run the system on my test sets. (If you tune parameters, improving accuracy as you go, the estimate still might be a bit high.)

I have posted a file on the course website called `TC_provided.tar.gz`. If you download this file to a Linux system or to a Windows system with Cygwin, you can extract the contents by typing "`gunzip TC_provided.tar.gz`" and then "`tar -xvf TC_provided.tar`". This will create a directory called `TC_provided`, including the following files:

- `corpus1_train.labels`: This file contains the list of labeled training files for corpus 1. Note that there are 885 training documents, and the 5 categories are not represented equally (there are 282 `Str`, 243 `Pol`, 207 `Dis`, 100 `Cri`, and 53 `Oth` documents).
- `corpus1_test.list`: This file contains the list of 443 test files for corpus 1.
- `corpus1_test.labels`: This file contains the list of labeled test files for corpus 1. I am providing this to you so that you can evaluate your text categorization system after applying it to the files listed in corpus1_test.list.
- `corpus2_train.labels`: This file contains the list of labeled training files for corpus 2. Note that there are 894 training documents including 621 with category `O` and 273 with category `I`.
- `corpus3_train.labels`: This file contains the list of labeled training files for corpus 3. Note that there are 955 training documents, and the 6 categories are not represented equally (there are 338 `Wor`, 245 `USN`, 124 `Sci`, 114 `Fin`, 92 `Spo`, and 42 `Ent` documents).

There will also be 3 subfolders called `corpus1`, `corpus2`, and `corpus3`. Within `corpus1`, you will find both the training and test documents. Within `corpus2` and `corpus3`, you will find only the training documents. Of course, I have test sets for all three corpora. In all cases, files were distributed randomly between the training and test sets, so while the relative sizes of categories in the test set will not be identical to the training set, you can expect them to be similar. In all three cases, approximately 1/3 of the documents were placed in the test set.

I have also provided, in the root directory, a file called `analyze.pl`, a Perl script I wrote (as a graduate student) that you can use to compare the output of your system to a file with the actual labels for the test set. You can use this directly to evaluate your system's results for corpus 1. For

example, if you name your output file `corpus1_predictions.labels` for corpus 1, you can type the following (assuming Perl is installed on your system): "`perl analyze.pl corpus1_predictions.labels corpus1_test.labels`"

If your output file is in the correct format, this will display a confusion matrix indicating the breakdown of correct and incorrect predictions according to categories, with columns indicating actual categories and rows indicating predictions. You will also see reported the precision, recall, and F1 measures for each category, as well as the overall accuracy of the system.

A user-friendly text categorization system should not assume specific categories; it should learn them from the training set that is provided. However, you may assume that your system will only be applied to the sets of categories mentioned in this document, and you may hardcode them into your system if you wish. In any case, you should not require the user to indicate which set of categories is being used. If you decide to hardcode the categories, your system should still figure out which of the three sets of categories it is dealing with based on the training set. (Of course, you also have the option of creating a general system which detects all the categories based on the training set.) Also, I don't think I should have to specify this, but *for the first corpus, you may not hardcode the answers for any specific document, nor may you include rules that are in any way specific to the examples in the test set*. Doing something like that would invalidate the evaluation for that corpus.

Your project may be written in any language, as long as I can run it easily using Cygwin or Ubuntu, but *I recommend using Python.* If you e-mail me your program early (at least two days before the deadline), I will test it on the three data sets and let you know the performance on the test sets (one of which is provided to you). You will then have an opportunity to improve your system and resubmit it if you are not satisfied. Expect that it might take me a couple of days to reply to each presubmission. Assuming that your program meets all the requirements, the grade will largely depend on how well the system performs, according to the overall accuracy metric. *I will specifically consider the performance relative to other systems I have received over the years that have used the same sort of machine learning approach*.

When you submit your project, I would also like *a short write-up* describing your system; one or at mots two pages should be enough. The information provided in your writeup should include:

- *Instructions explaining how to use your system*, including:
    - What operating system did you use to develop your system?
    - What programming language and version did you use?
    - How do I run your program (and how do I compile it, if necessary)?
    - What libraries do I need (and how can I install them if they are not already installed)?
- Which basic machine learning method did you use?
- How does your system tokenize training and test files?
- What weighting scheme, if any, is used for tokens (not relevant for naïve Bayes)?
- If you used naïve Bayes, what method of smoothing is used?

- Which optional parameters or features did you experiment with (e.g., possibilities might include case sensitivity, stemming or lemmatization, stop lists, etc.)? Which parameters or features made a significant difference, and how are they set in your final system?
- How did you evaluate your system's performance for the second and third data sets?
- You may include any additional information that you wish.

E-mail your final submission (and optionally presubmissions) to *CarlSable.Cooper@gmail.com*. NOTE: I am requesting that submissions be sent to my Cooper Gmail account because it is better for accepting Python programs. The project is due the night of Sunday, October 13, before midnight. I advise you to get started early and have fun with it!